1. Consider an algorithm which sorts an array of numbers. What is the Big-O running time of each of (i) Insertion Sort, (ii) Selection Sort, and (iii) Merge Sort in the following scenarios:
   (a) The array is sorted in ascending order.
   (b) The array is sorted, but backwards (ie: descending order).
2. Based on the previous question, consider which sorting algorithm you would use in part (a), and which you would use in part (b). Why?
3. If you knew nothing about the dataset (ie: you don't know if the array is mostly sorted or if it's totally unsorted), would you choose to write a Selection Sort or an Insertion Sort? Why?
4. Write a program that, given an array of integers, *ints*, computes:

$$\sum_{i=0}^{n} (ints[i])^2 - ints[i] + 2$$

   If n is the number of elements in *ints*, what is the Big-O running time of your algorithm?
5. Write a program that, given a number, n, creates an array where each index, i, holds the i^{th} digit of n. IE, given 12345, we would create the array: {1,2,3,4,5}
6. Write a program that, given two integer arrays, a and b, creates an array c which is a concatenated with b. IE, $a = \{1, 2, 3\}$, and $b = \{4, 5, 6\}$, $\therefore c = \{1, 2, 3, 4, 5, 6\}$. What's the Big-O running time of your algorithm?
7. Write a program that, given an array of Strings, does two passes of printing all of the strings (IE: if there are six strings, it will print all six strings once, and then print them all again). What is the Big-O running time of your algorithm?
8. When is it advantageous (read: *better*) to use Binary Search instead of Sequential Search?
   (a) If you only need to do one search and your data is unsorted, is it better to use Binary or Sequential Search?
   (b) If you only need to do one search and your data is sorted, is it better to use Binary or Sequential Search?
   (c) (Bonus) If you need to do a billion searches of your very large unsorted dataset, would it be better to do a billion Sequential Searches, or sort via MergeSort and then perform a billion Binary Searches?
9. Write a program that reverses a String, *str*, and prints the result to the console:
   (a) Using a loop.
   (b) Using recursion.

10. For each of the following, assume we have declared the following:

```
int[] arr = {1, 2, 3, 18};
int[] arr2 = new int[8];
int[][] arr3 = { {1, 2}, {3, 4}, {5, 6}, {7, 8}, {9, 10} };
String s = "Hello World";
```

Identify what the result will be, or write *Compile-time* or *Run-time* if there is a Compile-time or Run-time error:

(a) arr[0];
(b) arr[arr2.length - 5];
(c) s.charAt(arr.length);
(d) s.charAt(arr[3]);
(e) s.charAt(arr[5]);
(f) s.charAt(arr[1]);
(g) arr3[arr.length];
(h) arr3[arr.length][1];
(i) arr3.length;
(j) arr3[0].length;
(k) arr3[1].length();
(l) s.length;
(m) s.charAt(s.indexOf('H'));
(n) s.charAt(s.indexOf('Q'));
(o) s.indexOf('Q');
(p) arr2[0];
(q) arr2 = arr;
(r) arr[0] = arr3[2][0] + arr3[3][0];
(s) arr[0] = (int) s.charAt(6);
(t) arr[0] = s.charAt(6);
(u) int j = 6.0;
(v) double d = 7;
(w) s += 'q';
(x) (s + "Goodbye!").length();
(y) s == ("Hello " + "World");
(z) s.equals("Hello " + "World");

11. In Milestone 2, we asked you to write a Player class which had the following methods:

```
public void deal(Card c);
public Card[] discard();
public double wager(double min);
public Hand showHand();
public Hand showHand();
public double getBalance();
public void winnings(double amount);
```

Imagine a Poker game in which we have an array of players. Write a method that finds the total amount of money among all of the Players, and then computes the average amount of money held by a Player

$$\text{(Hint): average} = \sum_{i}^{N} \frac{\text{Player } i\text{'s balance}}{N}$$

```
public static double findAverageBalance(Players[] players){




}
```

Now, let's assume that for a given Player, their minimum bet to be able to play is defined as:

$$\begin{cases} \text{player's balance * } 10\%, & \text{player's balance} < \text{average table balance} \\ \text{player's balance * } 20\%, & \text{player's balance} \geq \text{average table balance} \end{cases}$$

Write a method to compute this minimum within the Player class:

```
public double getMinimumBalance(double averageTableBalance){




}
```

12. Here is an implementation of fibonacci:

```
public int fibonacci(int n){
   if(n == 0) return 1;
   if(n == 1) return 1;
   return fibonacci(n - 1) + fibonacci(n - 2);
}
```

(a) The running time of this algorithm is $O(2^n)$. Is that *better* or *worse* than an algorithm that runs in $O(n^2)$? Is $O(2^n)$ *better* or *worse* than an algorithm that runs in $O(n)$? How about one that runs in $O(log_2(n))$ time? Rank them, from *fastest* to *slowest*:

(b) Write a program that computes fibonacci of a given n, but without recursion:

```
public int fibonacci(int n){




}
```

(c) What's the running time of the algorithm you wrote?

13. Find at least five **syntactical** errors in the following program:

```
int j = 2.0;
answer = 0;
for(i = j; i < 12 i ++)
   answer = answer + Math.pow{i, 2);
}
```

14. The following code snippet is meant to find the sum of all of the numbers between 1 and 100. Find three **logical** errors:

```
int sum = 0;
for(int i = -1; i > 100; i ++){
   sum -= i;
}
```

15. Trace a MergeSort of the following dataset, showing the number of comparisons in each pass:

$$\{7, 12, 100, 18, 23, 8, 1, 3, 4, 0, 19, 6, 4, 1, 20, 12\}$$

16. Trace an Insertion Sort of the following dataset, showing the number of comparisons in each pass:

$$\{8, 1, 3, 4, 0, 19\}$$

17. Trace a Binary Search of the following dataset, where we are attempting to find the number 8.

$$\{1, 3, 5, 6, 12, 19, 20, 25, 28, 33\}$$